6、集合

6.1 常用对象

6.1.1 String、StringBuffer和StringBuilder

String类的特点：

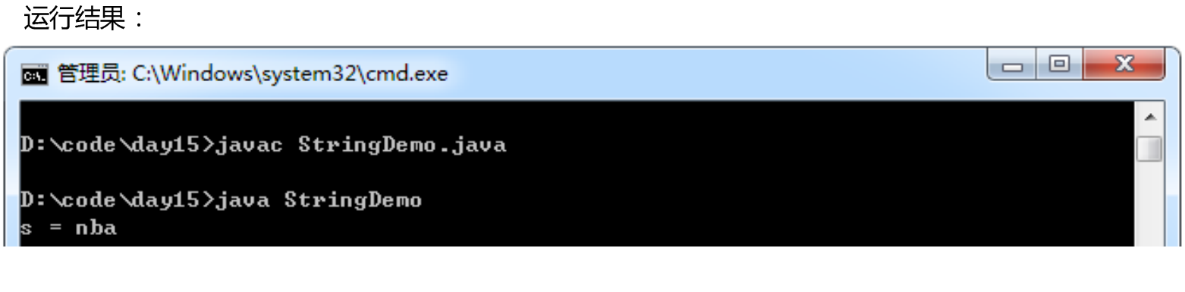
字符串对象一旦被初始化就不会被改变。

示例1：

```
public class StringDemo{
        public static void main(String[] args){
              String s = "abc";
              s = "nba";
              System.out.println("s = " + s);
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringDemo.java

D:\code\day15>java StringDemo
s = nba
```

原因分析：

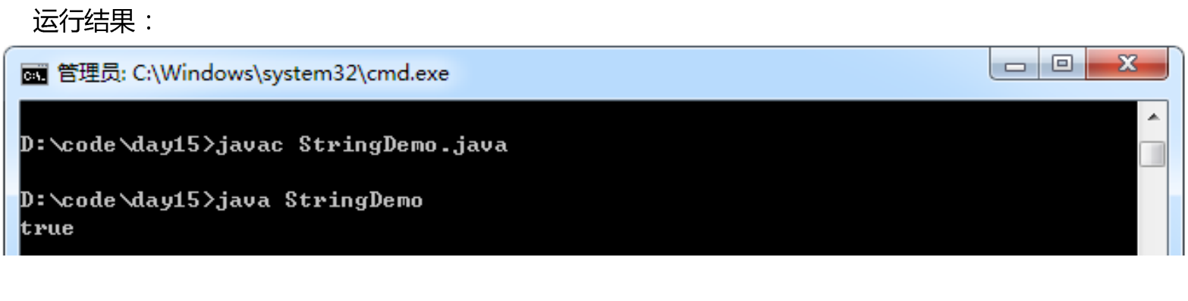
"abc"字符串对象并没有被改变，只是引用变量s指向了新创建的字符串对象"nba"。

示例2：

```
public class StringDemo{
        public static void main(String[] args){
              String s1 = "abc";
              String s2 = "abc";
              System.out.println(s1 == s2);
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringDemo.java

D:\code\day15>java StringDemo
true
```

原因分析：

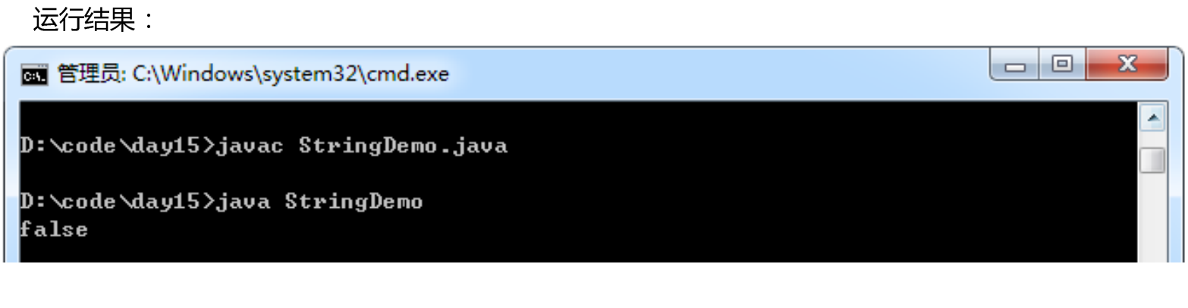
字符串创建的时候，有一个字符串常量池，s1创建后，"abc"放入其中。s2创建的时候，"abc"已经存在于字符串常量池中，故引用变量s2直接指向了已经存在的"abc"字符串对象，故s1==s2。

示例3：

```
public class StringDemo{
        public static void main(String[] args){
              String s1 = "abc";
              String s2 = new String("abc" );
              System.out.println(s1 == s2);
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringDemo.java

D:\code\day15>java StringDemo
false
```

原因分析：

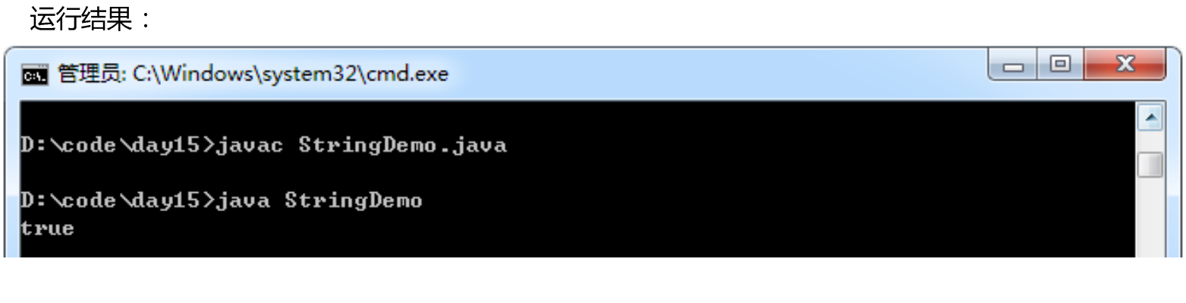
s1创建后，是在字符串常量池中创建了一个"abc"字符串对象。而s2是在堆内存中创建了另外一个"abc"字符串对象。所以，两个对象不是同一个对象。

示例4：

```
public class StringDemo{
        public static void main(String[] args){
              String s1 = "abc";
              String s2 = new String("abc" );
              System.out.println(s1.equals(s2));
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringDemo.java

D:\code\day15>java StringDemo
true
```

原因分析：

String类复写了Object中的equals方法，建立了String类自己的判断字符串对象是否相同的依据。只比较字符串内容，不比较地址。

String类的构造函数

构造函数：String(bytes[] bytes)

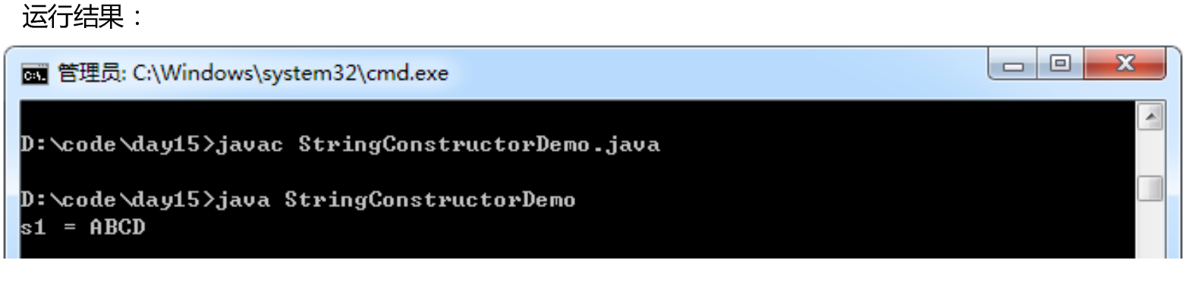
`String(byte[] bytes)`
通过使用平台的默认字符集解码指定的 byte 数组，构造一个新的 String。

示例1：

```
public class StringConstructorDemo {
     public static void main(String[] args){
          StringConstructorDemo();
     }

     public static void StringConstructorDemo(){
         String s = new String();//等效于String s = "";不等效于String s = null;

         byte[] arr = {65,66,67,68};
         String s1 = new String(arr);
         System.out.println("s1 = " + s1);
     }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringConstructorDemo.java

D:\code\day15>java StringConstructorDemo
s1 = ABCD
```

构造函数：String(bytes[] bytes)

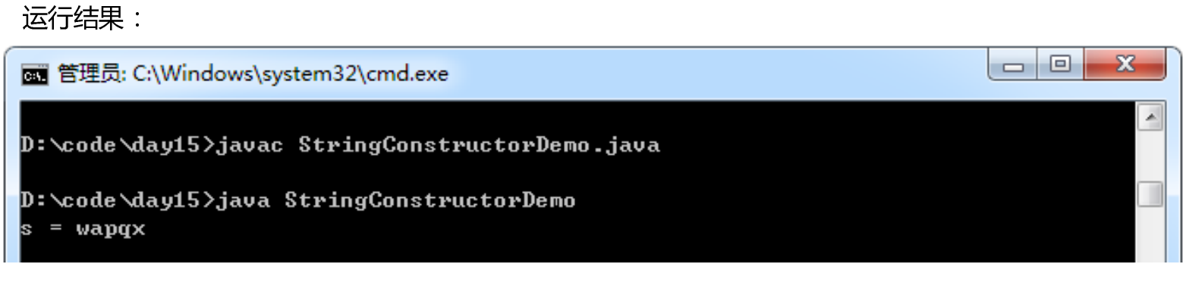
`String(char[] value)`
分配一个新的 String，使其表示字符数组参数中当前包含的字符序列。

示例2：

```
public class StringConstructorDemo {
        public static void main(String[] args){
               StringConstructorDemo();
        }

        public static void StringConstructorDemo(){
              char[] arr = {'w' ,'a' ,'p' ,'q' ,'x' };
              String s = new String(arr);
              System.out.println( "s = " + s);
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringConstructorDemo.java

D:\code\day15>java StringConstructorDemo
s = wapqx
```

构造函数：String(bytes[] bytes)

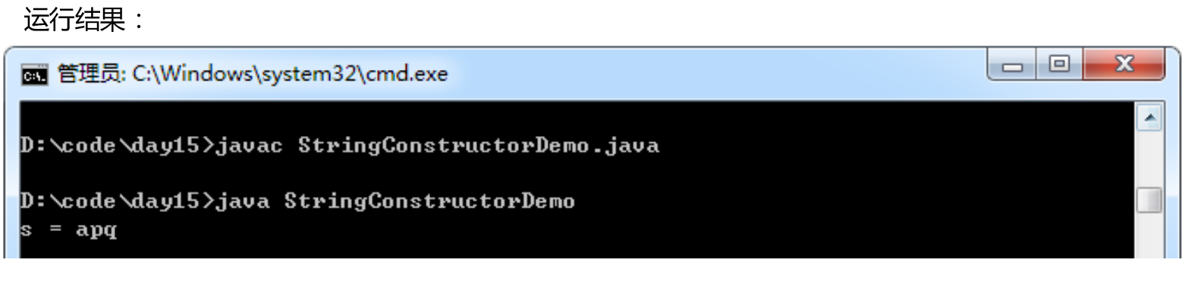
`String(char[] value, int offset, int count)`
分配一个新的 String，它包含取自字符数组参数一个子数组的字符。

示例3：

```
public class StringConstructorDemo{
        public static void main(String[] args){
              StringConstructorDemo();
        }

       public static void StringConstructorDemo(){
              char[] arr = {'w' ,'a' ,'p' ,'q' ,'x' };
              String s = new String(arr,1,3);
              System.out.println( "s = " + s);
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringConstructorDemo.java

D:\code\day15>java StringConstructorDemo
s = apq
```

String类部分方法

1、获取

获取字符串中字符的个数（长度）

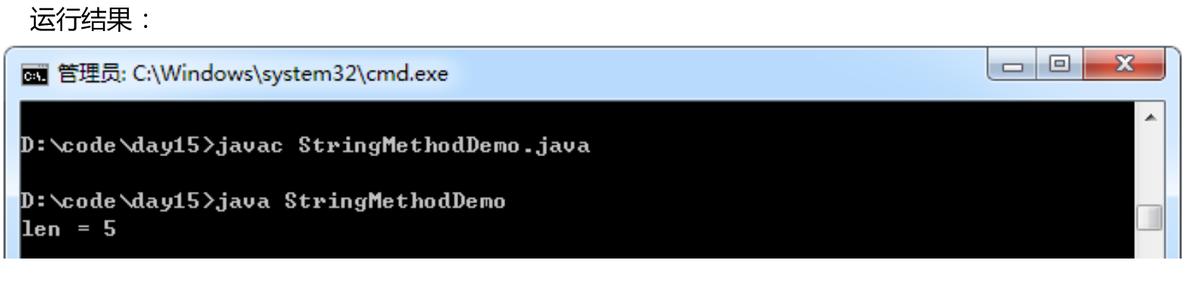
int length();

示例1：

```
public class StringMethodDemo {
        public static void main(String[] args){
               StringMethodDemo();
        }

       public static void StringMethodDemo() {
              String s = new String("abcde" );
              System.out.println( "len = " + s.length());
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
len = 5
```

根据位置获取字符

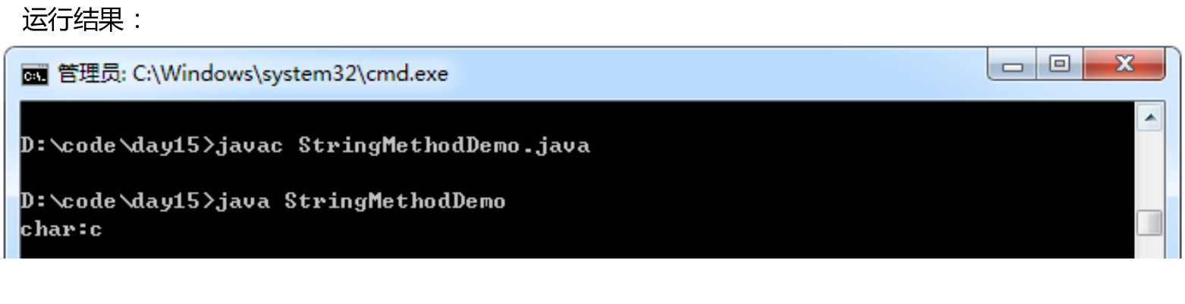
char charAt(int index);

示例2：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "abcde";
         System.out.println( "char:" + s.charAt(2));
     }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
char:c
```

根据字符获取在字符串中的位置

int indexOf(int ch);

indexO方法参数f类型为int是为了既可以支持字符，也可以支持字符在ASCII码中对应的数字。

从指定位置开始查找ch第一次出现的位置。

int indexOf(int ch,int fromIndex);

int indexOf(String str);

int indexOf(String str,int fromIndex);

根据字符串获取在字符串中第一次出现的位置。

int lastIndexOf(int ch);

int lastIndexOf(int ch,int fromIndex);

int lastIndexOf(String str);

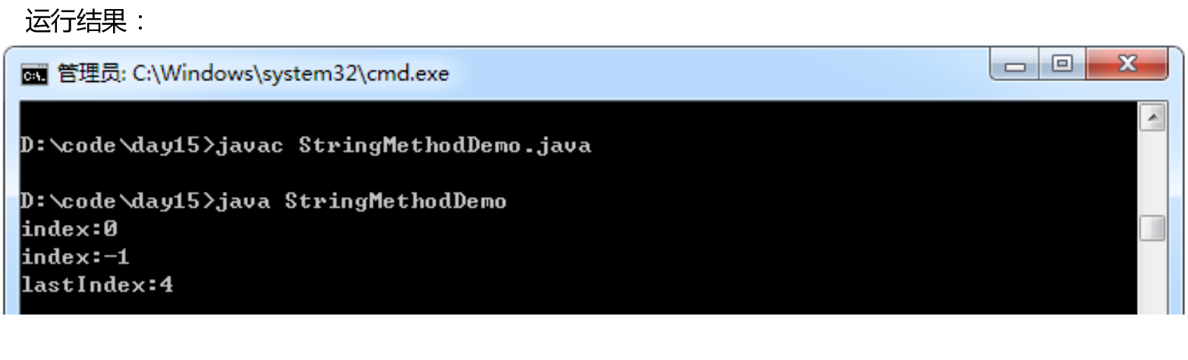
int lastIndexOf(String str,int fromIndex);

示例3：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "abcdae";
         System.out.println( "index:" + s.indexOf('a' ));
         System.out.println( "index:" + s.indexOf('k' ));
         System.out.println( "lastIndex:" + s.lastIndexOf('a' ));
     }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
index:0
index:-1
lastIndex:4
```

P.S.

可以根据-1，来判断该字符或者字符串是否存在。

获取字符串中的一部分字符串，也叫子串。

String substring(int beginIndex,int endIndex);

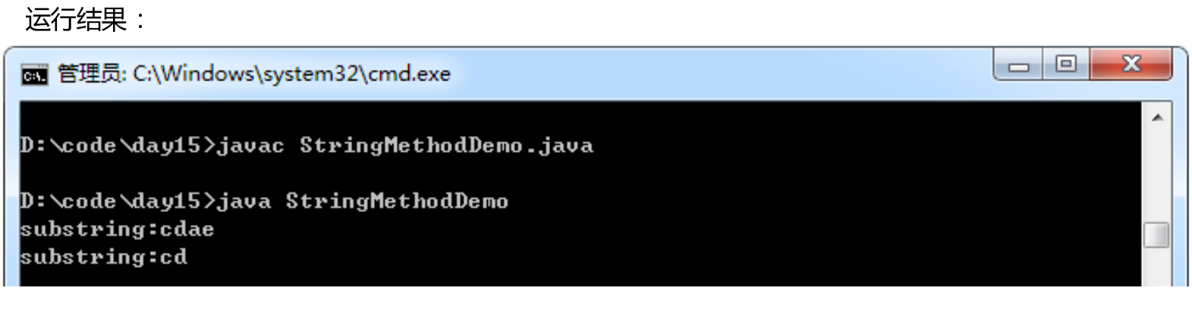
String substring(int beginIndex);

示例4：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "abcdae";
         System.out.println( "substring:" + s.substring(2));
         System.out.println( "substring:" + s.substring(2,4));
     }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
substring:cdae
substring:cd
```

2、转换

将字符串变成字符串数组（字符串的切割）

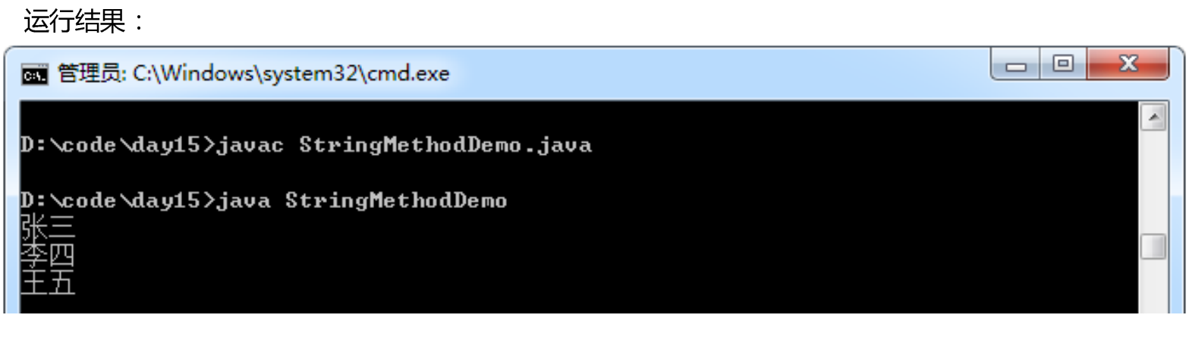
String[] split(String regex)；涉及到正则表达式。

示例5：

```
public class StringMethodDemo {
     public static void main(String[] args){
          StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "张三，李四，王五"；
         String[] arr = s.split( "，");

          for(int i = 0; i < arr.length; i++){
              System.out.println(arr[i]);
          }
     }
}
```

复制代码

运行结果：



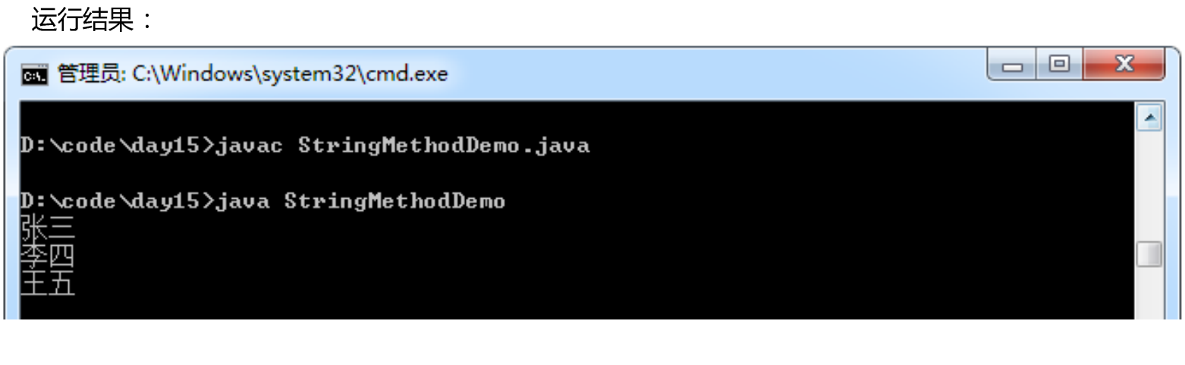
```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
张三
李四
王五
```

示例6：

```
public class StringMethodDemo{
        public static void main(String[] args){
              StringMethodDemo();
        }

       public static void StringMethodDemo(){
              String s = "张三.李四.王五" ;
              String[] arr = s.split( "\\.");

               for(int i = 0; i < arr.length; i++){
                    System.out.println(arr[i]);
               }
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
张三
李四
王五
```

P.S.

点在正则表达式中是特殊符号，需要转义。

将字符串变成字符数组

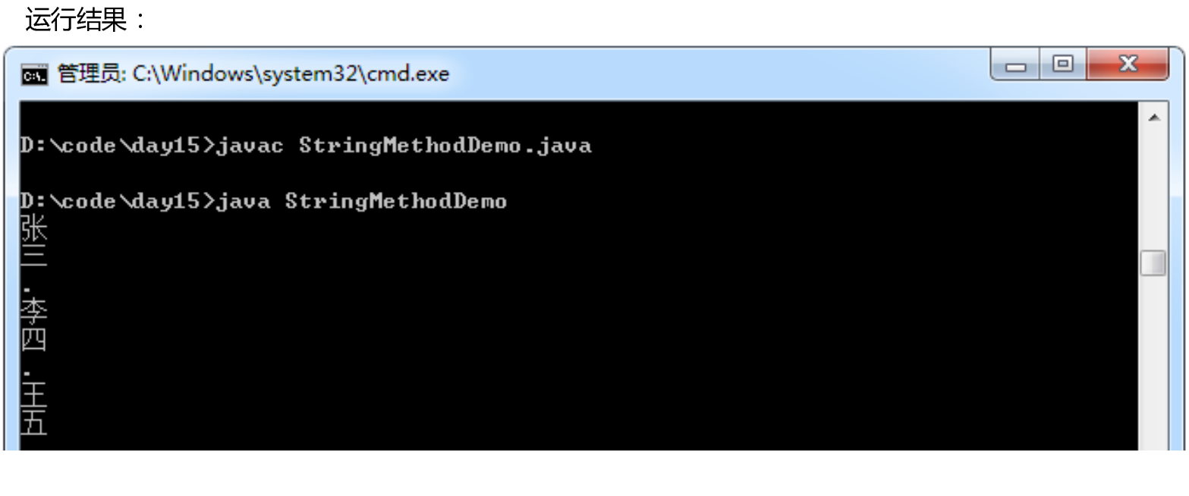
char[] toCharArray();

示例7：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "张三.李四.王五"；
          char[] chs = s.toCharArray();

          for(int i = 0; i < chs.length; i++){
              System.out.println(chs[i]);
          }
     }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
张
三
.
李
四
.
王
五
```

将字符串变成字节数组

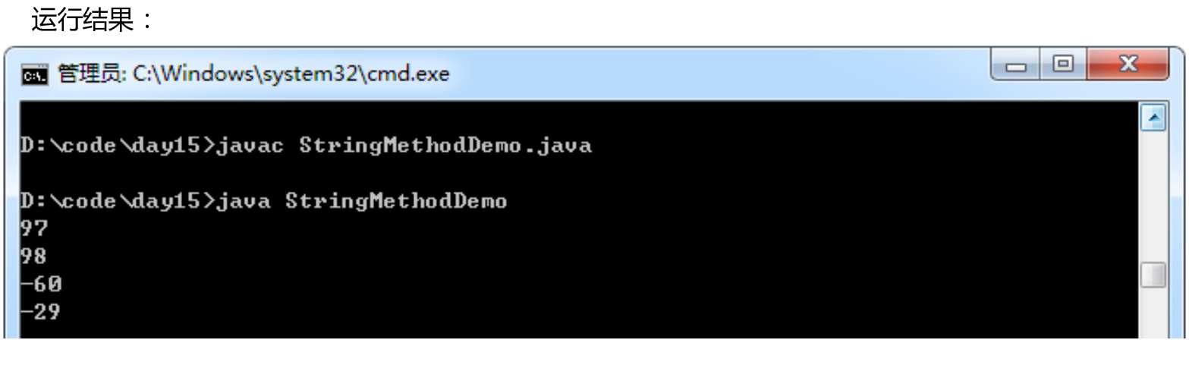
char[] getBytes();

示例8：

```
public class StringMethodDemo{
        public static void main(String[] args){
              StringMethodDemo();
        }

        public static void StringMethodDemo(){
              String s = "ab你" ;
               byte [] bytes = s.getBytes();

               for (int i = 0; i < bytes.length; i++){
                    System.out.println(bytes[i]);
               }
        }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
97
98
-60
-29
```

将字符串中的字母转成大小写

String toUpperCase();大写

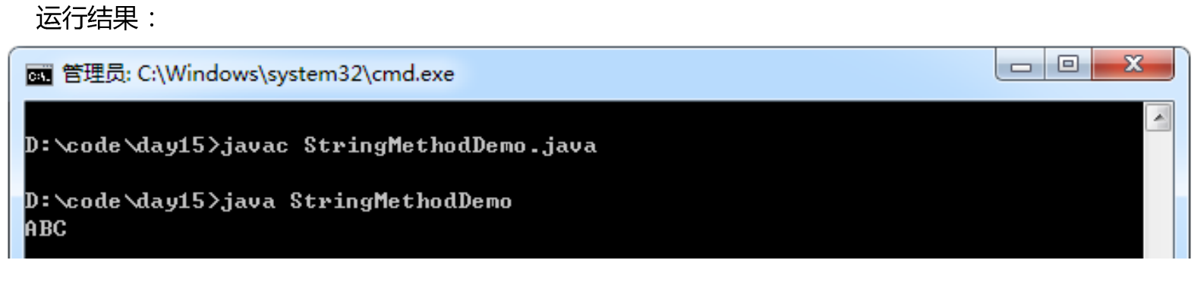
String toLowerCase();小写

示例9：

```
public class StringMethodDemo{
        public static void main(String[] args){
              StringMethodDemo();
        }

```

```
        public static void StringMethodDemo(){
             System.out.println( "Abc".toUpperCase());
        }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
ABC
```

将字符串中的内容进行替换

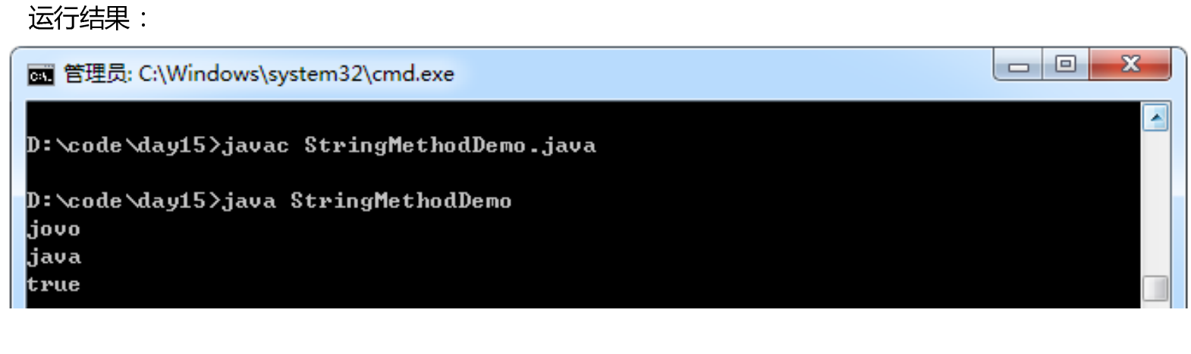
String replace(char oldCh,char newCh);

String replace(String s1,String s2);

示例10：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s1 = "java";
         String s2 = s1.replace( 'a','o' );
         String s3 = s1.replace( 'q','o' );

         System.out.println(s2);
         System.out.println(s3);
         System.out.println(s1 == s3);
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
jovo
java
true
```

P.S.

replace方法如果没有找到要替换的内容，则返回的还是原字符串。

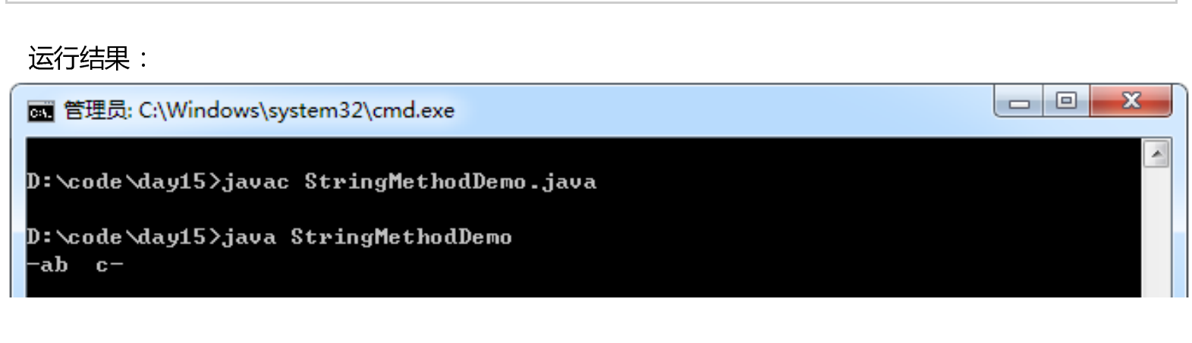
去除字符串两端空格

String trim();

示例11：

```
public class StringMethodDemo {
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         System.out.println( "-" + "   ab  c    " .trim() + "-");
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
-ab  c-
```

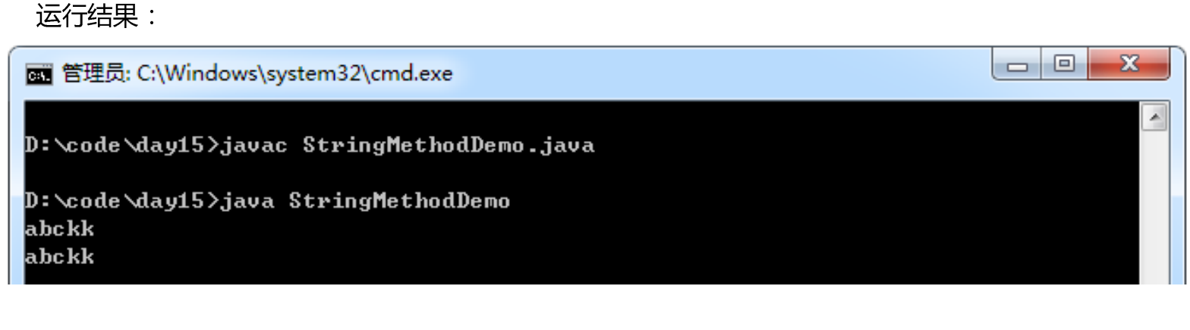
将字符串进行连接

String concat(String str);

示例12：

```
public class StringMethodDemo{
        public static void main(String[] args){
             StringMethodDemo();
        }

        public static void StringMethodDemo(){
             System.out.println( "abc".concat("kk" ));
             System.out.println( "abc" + "kk" );
        }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
abckk
abckk
```

P.S.

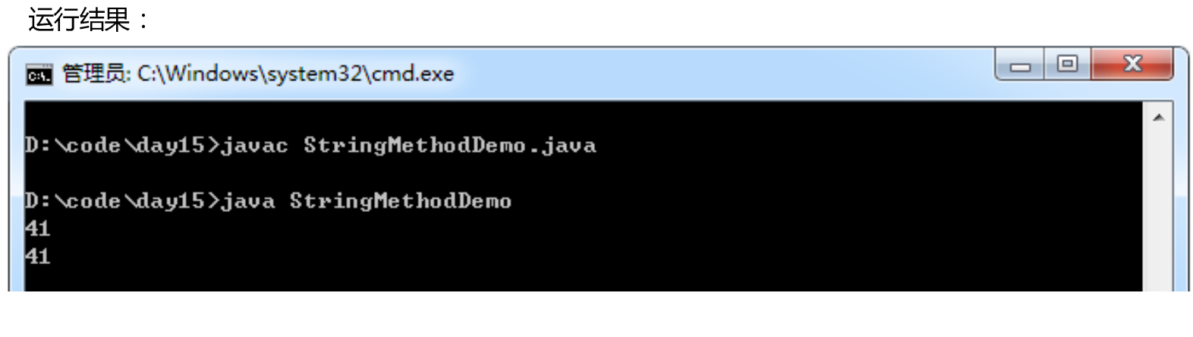
concat效果与"+"连接符效果一致，但是效果更高一些。

将其他类型数据转换成字符串

示例13：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         System.out.println(String.valueOf(4) + 1);
         System.out.println( "" + 4 + 1);
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
41
41
```

3、判断

两个字符串内容是否相同呢？

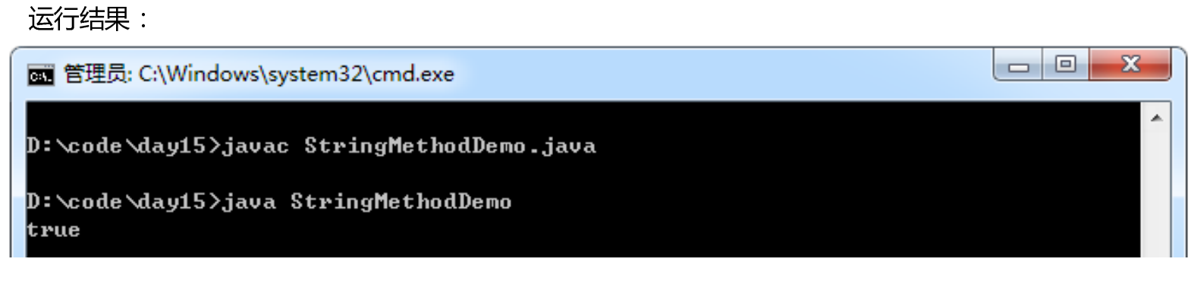
boolean equals(Object obj);

boolean equalsIgnoreCase(String str);忽略大小写比较字符串内容。

示例14：

```
public class StringMethodDemo {
     public static void main(String[] args){
          StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s = "abc";
         System.out.println(s.equalsIgnoreCase("ABC" ));
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
true
```

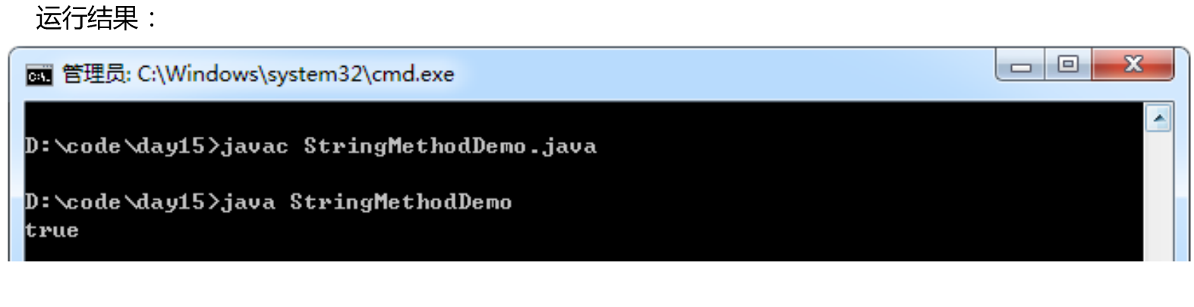
字符串中是否包含指定字符串

boolean contains(String str);

示例15：

```
public class StringMethodDemo{
        public static void main(String[] args){
             StringMethodDemo();
        }

        public static void StringMethodDemo(){
             String s = "abc";
             System.out.println(s.contains( "bc"));
        }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
true
```

3.3 字符串是否以指定字符串开头，是否以指定字符串结尾

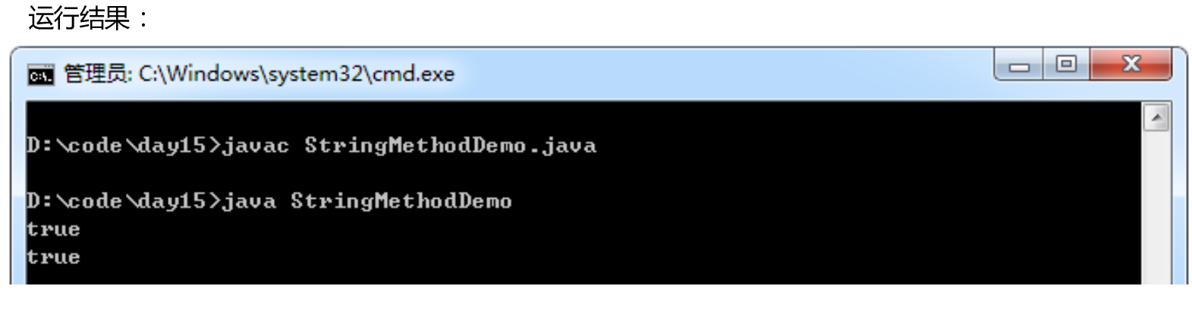
boolean startsWith(String str);

boolean endsWith(String str);

示例16：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String str = "ArrayDemo.java";
         System.out.println(str.startsWith( "Array"));
         System.out.println(str.endsWith( "java"));
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
true
true
```

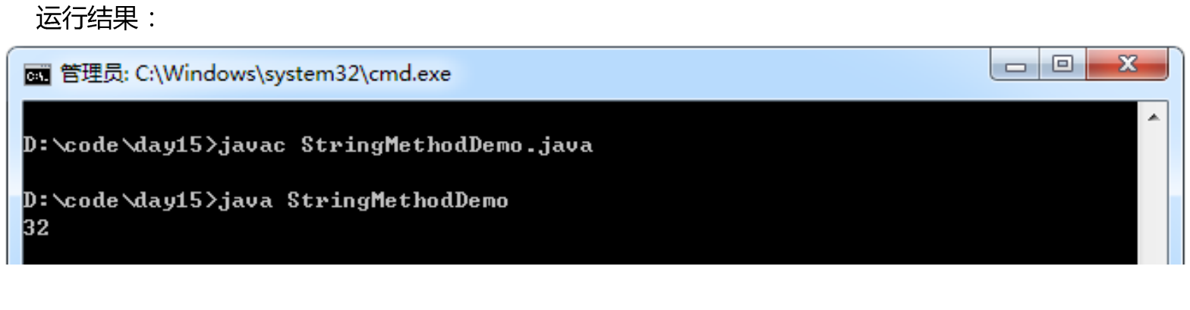
4、比较

int compareTo(String str);   如果参数字符串等于此字符串，则返回值0；如果此字符串按字典顺序小于字符串参数，则返回一个小于0的值；如果此字符串按字典顺序大于字符串参数，则返回一个大于0的值。

示例17：

```
public class StringMethodDemo {
        public static void main(String[] args){
              StringMethodDemo();
        }

        public static void StringMethodDemo(){
             System. out.println("a" .compareTo("A"));
        }
}
```
复制代码

运行结果：

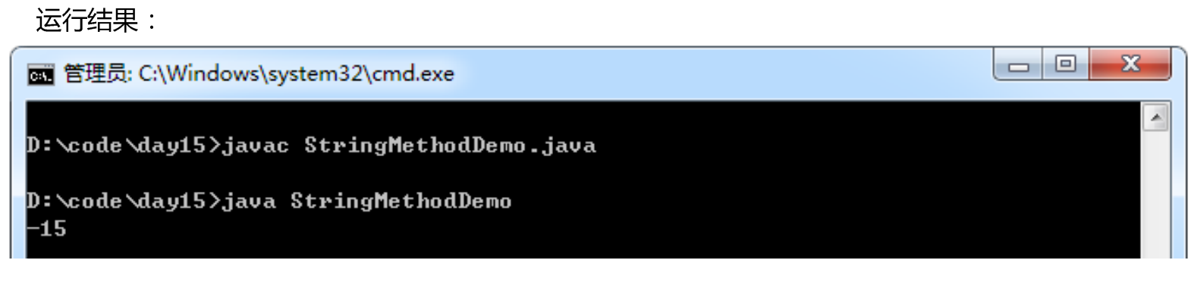


```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
32
```

示例18：

```
public class StringMethodDemo{
        public static void main(String[] args){
             StringMethodDemo();
        }

        public static void StringMethodDemo(){
             System.out.println( "abc".compareTo("aqz" ));
        }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
-15
```

P.S.

"abc"和"aqz"两个字符串比较，'a'和'a'相等，'b' - 'q' = -15，'c'和'z'也就没有必要比较。

5、返回字符串对象的规范化表示形式

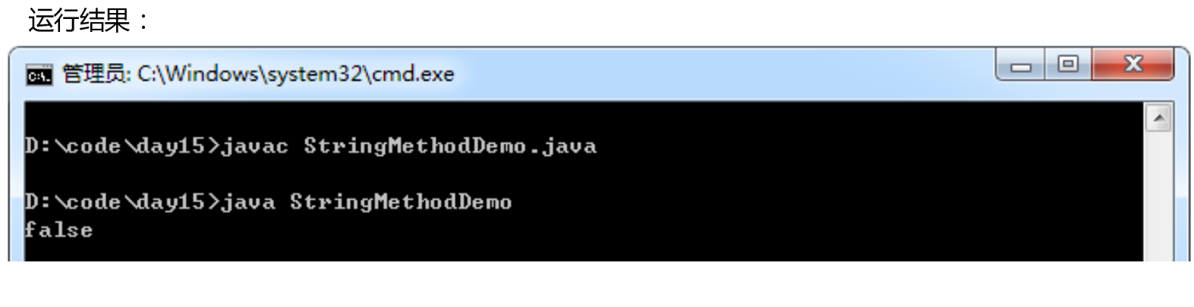
String intern();

当调用intern方法时，如果池已经包含一个等于此String对象的字符串（用equals(Object)方法确定），则返回池中的字符串。否则，将此String对象添加到池中，并返回此String对象的引用。

示例19：

```
public class StringMethodDemo{
     public static void main(String[] args){
         StringMethodDemo();
     }

     public static void StringMethodDemo(){
         String s1 = new String("abc" );
         String s2 = s1.intern();

         System.out.println(s1 == s2);
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringMethodDemo.java

D:\code\day15>java StringMethodDemo
false
```

练习

1、给定一个字符串数组，按照字典顺序进行从小到大的排序。

{"nba","abc","cba","zz","qq","haha"}

思路：

①对数组排序，可以用选择，冒泡都行。

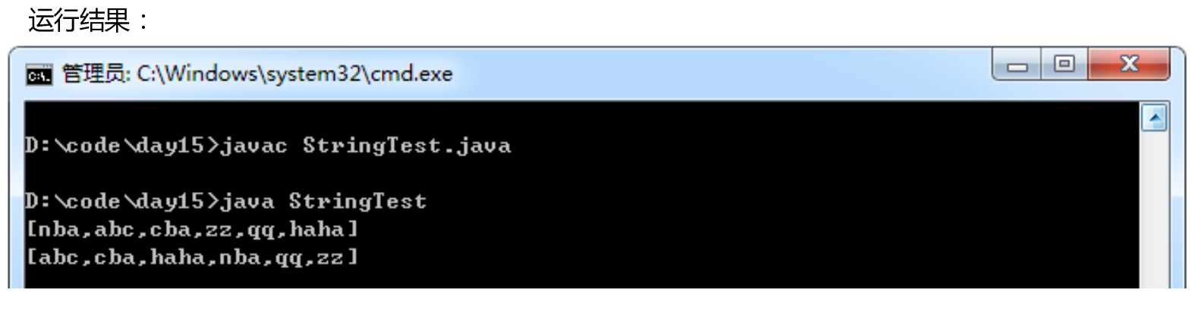
②for嵌套、比较以及换位。

③使用String类中提供的用于字符串比较的功能进行比较。

代码：

```
public class StringTest{
     public static void main(String[] args){
         String[] arr = { "nba","abc" ,"cba" ,"zz" ,"qq" ,"haha" };

         printArray(arr);
         sortString(arr);
         printArray(arr);
     }

     public static void printArray(String[] arr){
          System.out.print( "[");
          for(int i = 0; i < arr.length; i++){
               if(i != arr.length -1)
                    System.out.print(arr[i] + ",");
               else
                    System.out.println(arr[i] + "]");
          }
     }

     public static void sortString(String[] arr){
          for(int i = 0; i < arr.length - 1; i++){
               for(int j = i + 1; j < arr.length; j++){
                    if(arr[i].compareTo(arr[j])>0){
                         swap(arr,i,j);
                    }
               }
          }
     }

     private static void swap(String arr[],int i, int j){
         String temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringTest.java

D:\code\day15>java StringTest
[nba,abc,cba,zz,qq,haha]
[abc,cba,haha,nba,qq,zz]
```

2、一个子串在整串中出现的次数   "nbaernbatynbauinbaopnba"

思路：

①要找的子串是否存在，如果存在获取其出现的位置，这个可以使用indexOf完成。

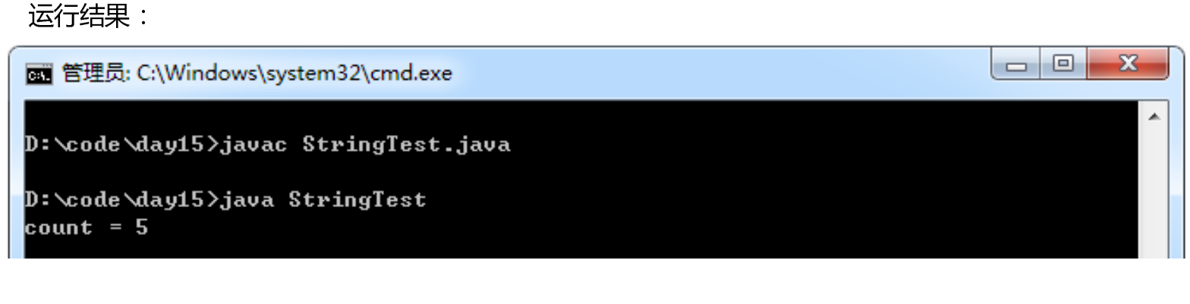
②如果找到了，那么就记录出现的位置并在剩余的字符串中继续查找该子串，而剩余字符串的起始位是出现位置+子串的长度。

③以此类推，通过循环完成查找，如果找不到就是-1，并且每次找到用计数器记录。

代码：

```
public class StringTest{
     public static void main(String[] args){
         String str = "nbaernbatynbauinbaopnba";
         String key = "nba";

          int count = getKeyStringCount(str,key);
          System.out.println( "count = " + count);
     }

    public static int getKeyStringCount(String str,String key){
          //1、定义计数器
          int count = 0;

          //2、定义变量记录key出现的位置
          int index = 0;

          while((index = str.indexOf(key)) != -1){
              str = str.substring(index + key.length());
              count++;
          }

          return count;
     }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringTest.java

D:\code\day15>java StringTest
count = 5
```

3、两个字符串中最大相同的子串

"qwerabcdtyuiop"

"xcabcdvbn"

思路：

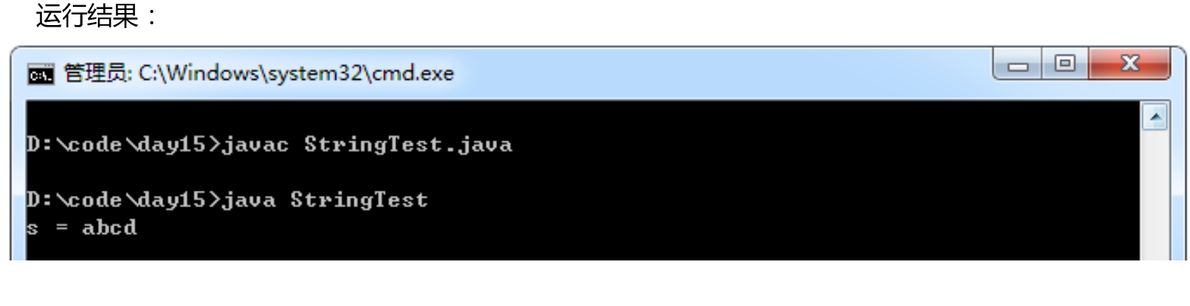
①既然取的是最大子串，先看短的那个字符串是否在长的那个字符串中。如果存在，短的那个字符串就是最大子串。

②如果不是，那么就将短的那个子串进行长度递减的方式去子串，去子串中判断是否存在。如果存在就已找到，就不用再找了。

代码：

```
public class StringTest{
       public static void main(String[] args){
            String s1 = "qwerabcdtyuiop";
            String s2 = "xcabcdvbn";
            String s = getMaxSubstring(s1,s2);
            System.out.println( "s = " + s);
      }

       public static String getMaxSubstring(String s1,String s2){

            String max = null,min = null;
            max = (s1.length() > s2.length())?s1:s2;
            min = max.equals(s1)?s2:s1;

             for(int i = 0; i < min.length(); i++){
                   for(int a = 0,b = min.length() - i; b != min.length() + 1;
a++,b++){
                        String sub = min.substring(a,b);
                         if(max.contains(sub))
                               return sub;
                  }
            }
             return null ;
      }
}
```
复制代码

运行结果：



```
D:\code\day15>javac StringTest.java

D:\code\day15>java StringTest
s = abcd
```

4、模拟一个trim功能方法，去除字符串两端的空白。

思路：

①定义两个变量。

一个变量作为从头开始判断字符串空格的角标，不断++。

一个变量作为从尾开始判断字符串空格的角标，不断--。

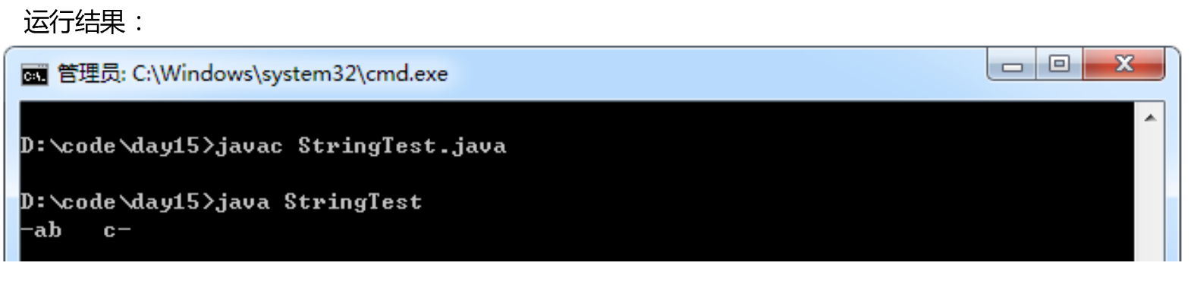
②判断到不是空格为止，取头部之间的字符串即可。

代码：

```
public class StringTest{
      public static void main(String[] args){
          String s = "   ab   c    " ;

          s = myTrim(s);
          System.out.println( "-" + s + "-" );
      }

      public static String myTrim(String s){
           int start = 0,end = s.length() - 1;

           while(start <= end && s.charAt(start) == ' '){
               start++;
           }

           while(start <= end && s.charAt(end) == ' '){
               end--;
           }
           return s.substring(start,end + 1);
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringTest.java

D:\code\day15>java StringTest
-ab   c-
```

StringBuffer

StringBuffer：就是字符串缓冲区，用于存储数据的容器。

特点：

1. 长度是可变的。
2. 可以存储不同类型数据。
3. 最终要转成字符串进行使用。

P.S.

StringBuffer的字符串缓冲区初始容量为16个字符，其实质还是数组。

StringBuffer既然是一个容器对象，应该具备什么功能呢？

1、添加

StringBuffer append(data);

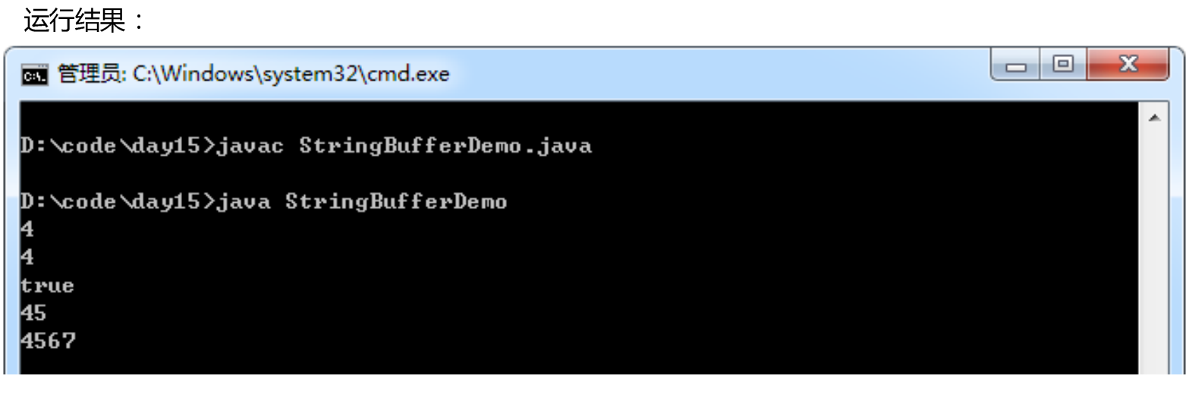
StringBuffer insert(index,data);

示例：

```
public class StringBufferDemo{
       public static void main(String[] args){
            bufferMethodDemo();
      }

       public static void bufferMethodDemo(){
            //创建缓冲区对象
            StringBuffer sb1 = new StringBuffer();
            StringBuffer sb2 = sb1.append(4);

            System.out.println(sb1);
            System.out.println(sb2);
            System.out.println(sb1 == sb2);

            System.out.println(sb1.append(5));
            System.out.println(sb1.append(6).append(7));
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBufferDemo.java

D:\code\day15>java StringBufferDemo
4
4
true
45
4567
```

原因分析：

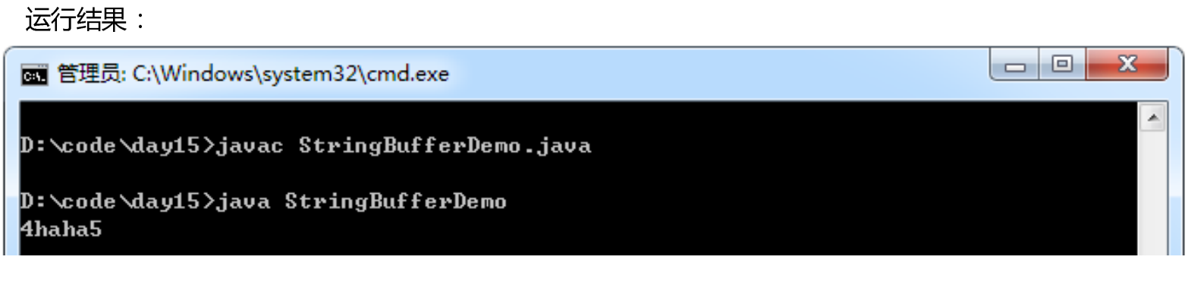
sb1.append(4);语句是在缓冲区中添加4，然后将sb2引用变量指向了最终生成的字符串对象，sb1也指向这个新生成的字符串对象，故两者指向的是同一个对象。

示例：

```
public class StringBufferDemo{
       public static void main(String[] args){
            bufferMethodDemo();
      }

       public static void bufferMethodDemo(){
            StringBuffer sb = new StringBuffer();
            sb.append(4).append(5);
            sb.insert(1, "haha");
            System.out.println(sb.toString());
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBufferDemo.java

D:\code\day15>java StringBufferDemo
4haha5
```

P.S.

insert方法也可以对字符串进行修改。

2、删除

StringBuffer delete(int start,int end);包含头，不包含尾。

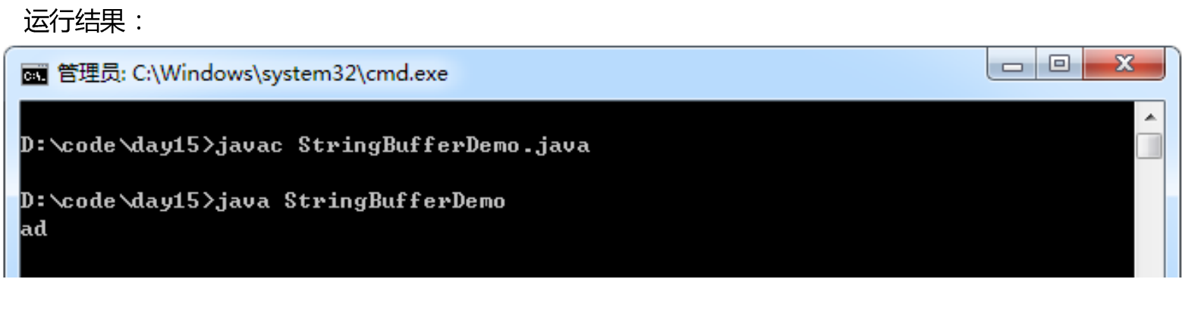
StringBuffer deleteCharAt(int index):删除指定位置的元素。

示例：

```
public class StringBufferDemo {
       public static void main(String[] args){
            bufferMethodDemo();
      }

       public static void bufferMethodDemo(){
            StringBuffer sb = new StringBuffer("abcd" );
            sb.delete(1,3);
            System.out.println(sb);

             //清空缓冲区
            sb.delete(0,sb.length());
            System.out.println(sb);
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBufferDemo.java

D:\code\day15>java StringBufferDemo
ad
```

3、查找

char charAt(int index);

int indexO(String str);

int lastIndexOf(String str);

4、修改

StringBuffer replace(int start,int end,String str);

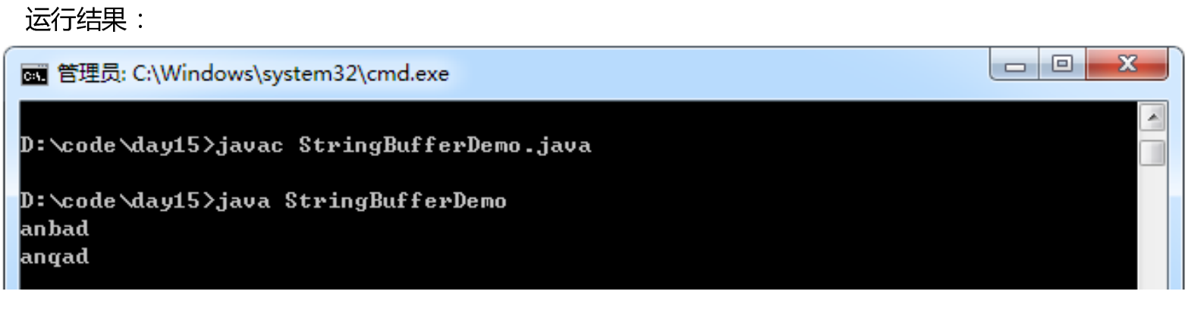
void setCharAt(int index,char ch);

示例：

```
public class StringBufferDemo{
       public static void main(String[] args){
            bufferMethodDemo();
      }

       public static void bufferMethodDemo(){
            StringBuffer sb = new StringBuffer("abcd" );

            sb.replace(1,3,"nba");
            System.out.println(sb);

            sb.setCharAt(2, 'q');
            System.out.println(sb);
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBufferDemo.java

D:\code\day15>java StringBufferDemo
anbad
anqad
```

5、其他方法

void setLength(int newLength);设置字符序列的长度

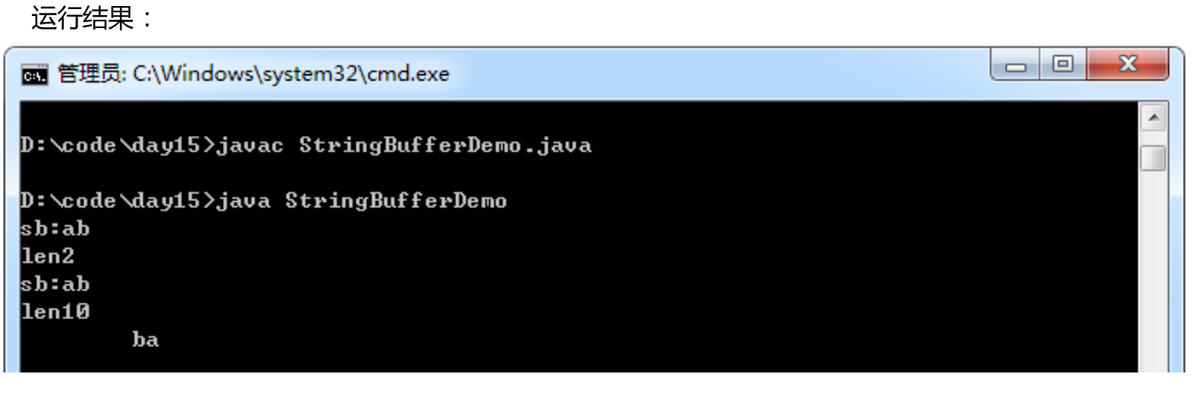
public StringBuffer reverse();将字符序列用其反转形式取代

示例：

```
public class StringBufferDemo{
       public static void main(String[] args){
            bufferMethodDemo();
      }

      public static void bufferMethodDemo(){
            StringBuffer sb = new StringBuffer("abcd" );

            sb.setLength(2);
            System.out.println( "sb:" + sb);
            System.out.println( "len" + sb.length());

            sb.setLength(10);
            System.out.println( "sb:" + sb);
            System.out.println( "len" + sb.length());

            System.out.println(sb.reverse());
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBufferDemo.java

D:\code\day15>java StringBufferDemo
sb:ab
len2
sb:ab
len10
        ba
```

P.S.

1、使用setLength设置StringBuffer中字符序列的长度。

如果小于已有字符序列的长度，相当于清除缓冲区中的一部分内容。

如果大于已有字符序列的长度，相当于扩充缓冲区，扩充部门内容用空格字符填充。

2、当创建的StringBuffer内容长度大于16，将会新创建一个新数组，长度比旧数组要长。然后把就数组的内容拷贝到新的数组，超出旧数组长度范围的内容将会放在新数组现在内容的后面，也可以通过StringBuffer(int capacity);构造函数自己设置StringBuffer缓冲区长度。

StringBuilder

jdk1.5以后出现了功能和StringBuffer一模一样的对象，就是StringBuilder。

不同的是：

StringBuffer是线程同步的，通常用于多线程。

StringBuilder是线程不同步的，通常用于单线程，它的出现能够提高程序效率。

故StringBuilder多用于多个线程是不安全的，如果需要这样的同步，则建议使用StringBuffer。

P.S.

JDK一般升级目的：

1、简化书写。

2、提高效率。

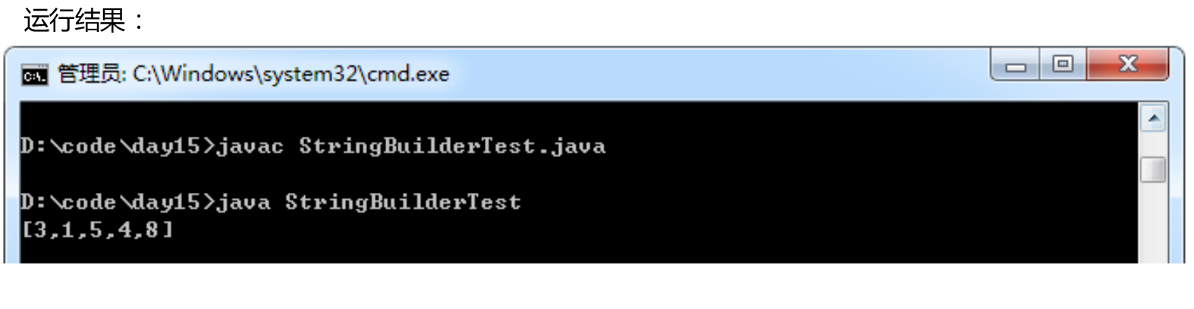
3、增加安全性。

示例：将一个int数组变成字符串。

```
class StringBuilderTest{
       public static void main(String[] args){
            int[] arr = {3,1,5,4,8};
            String s = arrayToString(arr);

            System.out.println(s);
      }

       public static String arrayToString(int[] arr){
            StringBuilder sb = new StringBuilder();
            sb.append( "[");
             for(int i = 0; i < arr.length; i++){
                   if(i != arr.length - 1)
                        sb.append(arr[i]).append( ",");
                   else
                        sb.append(arr[i]).append( "]");
            }
             return sb.toString();
      }
}
```

复制代码

运行结果：



```
D:\code\day15>javac StringBuilderTest.java

D:\code\day15>java StringBuilderTest
[3,1,5,4,8]
```

~END~

~爱上海，爱黑马~